Dominance Programming for Itemset Mining

Benjamin Negrevergne*, Anton Dries*, Tias Guns*, Siegfried Nijssen*†
*Department of Computer Science, KU Leuven, Belgium
†LIACS, Universiteit Leiden, The Netherlands
Email: {firstname.lastname}@cs.kuleuven.be

***Abstract*—Finding small sets of interesting patterns is an important challenge in pattern mining. In this paper, we argue that several well-known approaches that address this challenge are based on performing pairwise comparisons between patterns. Examples include finding closed patterns, free patterns, relevant subgroups and skyline patterns. Although progress has been made on each of these individual problems, a generic approach for solving these problems (and more) is still lacking. This paper tackles this challenge. It proposes a novel, generic approach for handling pattern mining problems that involve pairwise comparisons between patterns. Our key contributions are the following. First, we propose a novel algebra for *programming* pattern mining problems. This algebra extends relational algebras in a novel way towards pattern mining. It allows for the generic combination of *constraints* on individual patterns with *dominance relations* between patterns. Second, we introduce a modified generic constraint satisfaction system to evaluate these algebraic expressions. Experiments show that this generic approach can indeed effectively identify patterns expressed in the algebra.**

I. Introduction

Pattern mining constitutes a well established class of tasks in data mining. The most well-known task is *frequent* itemset mining, which consists in finding all sets of items that have a high support in a given transactional database. Unfortunately, basic frequent itemset mining is not very useful. The number of frequent itemsets is huge in many databases, even when using high support thresholds.

A large body of work has sought to solve this *pattern explosion* by mining for patterns under constraints. Constraint-based pattern mining is concerned with finding all patterns π in a pattern language $\mathcal{L}$ that satisfy some constraint φ [1]:

$$\text{Th}(\mathcal{L}, \mathcal{D}, \varphi) = \{X \in \mathcal{L} \,|\, \varphi(X, \mathcal{D}) \text{ is true}\}. \quad (1)$$

The constraint φ is typically a conjunction of multiple constraints that can be defined on the pattern X or the data $\mathcal{D}$. Given that φ is evaluated on patterns individually, φ is often called a *local constraint* [2]. Constraints usually come from domain-specific insights or are provided by the user to remove uninteresting patterns.

For example, one may require that the size of a pattern is smaller than some threshold, or that it does (not) contain certain items. One can also require that patterns have a high utility [3], that they satisfy syntactical constraints [4], or that they individually score well with respect to a given statistical test [5].

Numerous approaches to pattern mining have been developed to effectively find the patterns adhering to a set of local constraints.

The benefit of a constraint-based framework is twofold. First, users can combine existing constraints to formulate new problems according to their needs. Second, researchers can identify classes of constraints with similar properties and focus their attention on devising generic pruning strategies for these classes of constraints [6]. In particular, recent work has shown that *constraint programming* provides a generic framework to capture many pattern mining settings [7].

However the constraint-based pattern mining framework has limitations: several types of mining tasks cannot be formulated using constraints over individual patterns.

Let us consider the problem of *relevant pattern mining* [8] (or *subgroup discovery* [9]) as an example. This is a pattern mining task in a supervised setting where two databases are given (referred to as "positive" and "negative"). Mining relevant patterns consists in finding patterns that discriminate the positive dataset (D^+) from the negative one (D^-). A pattern P_1 that occurs in positive examples T_1^+ and negative examples T_1^- can be considered *irrelevant* in this setting if there is another pattern P_2 that occurs in positive examples T_2^+ and negative examples T_2^-, for which:

$$T_1^+ \subseteq T_2^+ \text{ and } T_2^- \subset T_1^-.$$

Since P_2 discriminates the two datasets better than P_1, we would like to specify that P_2 is a *better* solution than P_1 and that P_1 is a solution only if P_2 is not. Clearly, local constraints are inadequate for this purpose because they consider patterns individually.

Relevant pattern mining is not an isolated case. Many other pattern mining settings can not be formalized adequately using conjunctions of constraints. Bonchi and Lucchese [10] have shown that combinations of closedness and monotonic constraints such as the max-cost constraint can lead to ambiguous problem definitions in the constraint-based mining framework. Moreover, Crémilleux et al. [2] highlighted the need to use *global constraints* on patterns (i.e. constraints whose satisfaction depends on more than one pattern) to address problems such as finding condensed representations of patterns or top-k sets of patterns.

The **main insight** of this paper is that these settings can be formulated using a combination of constraints and *dominance relations*. Dominance relations are pairwise preferences between patterns. They can be used to express the idea that a pattern P_1 is preferred over another pattern P_2, or, in our terminology, that P_1 *dominates* P_2.

Building on this observation we introduce a unified algebra that can express both *constraints* and *dominance relations*. A key component of this framework is the *dominance algebra*, which allows to compose pairwise dominance relations into complex pre-orders among the patterns. We will show that many settings in the pattern mining literature can be formulated elegantly using compositions of constraints and dominance relations. Here, we do not only consider relevant subgroups, but also maximal patterns [11], closed [12] and free patterns [13] with any type of local constraint, sky patterns [14], dominated patterns in PN spaces [15], as well as new settings.

Another important contribution of this paper is that we demonstrate that this framework does not only provide a uniform approach to **formulate** these tasks, but also leads to a generic and effective method to **search** for solutions. Indeed, both local constraints and dominance relations can be used in a generic way to prune the search space. The expressions in this algebra can be evaluated effectively using a modified version of a constraint programming system [7]. Our experiments demonstrate that the resulting system exploits the dominance relations effectively and performs better than naive approaches, and in several cases, even better than specialized algorithms.

The paper is organized as follows: Section II gives an overview of several well-known pattern mining settings and the basic principles of dominance programming; Section III introduces the unified algebra to describe constraints and dominance relations. Section IV provides examples of problems expressed in the dominance algebra. Section V describes a method for evaluating expressions in this algebra based on constraint programming technology. Section VI evaluates the approach experimentally.

II. Dominance Relations in Itemset Mining

In this section, we introduce several well-known itemset mining settings, and demonstrate how the principle of dominance programming can be used to formulate these settings.

The itemset mining problem can be defined as follows. Let $I = \{1, \ldots, n\}$ be a set of items and $T = \{1, \ldots, m\}$ a set of transaction identifiers. A dataset is a set $D \subseteq I \times T$. The *cover of an itemset* is defined as:

$$cover_D(X) = \{t \in T : \forall i \in X, (i,t) \in D\} \tag{2}$$

and contains the identifiers of transactions in which all items of X occur.

Frequent itemset mining consists in enumerating all the subsets X of I whose cover is larger than a user defined minimum frequency threshold θ:

$$\text{Th}_{\text{fi}} = \text{Th}(I, \mathcal{D}, p) = \{X \subseteq I \,|\, |cover_D(X)| \geq \theta\}. \tag{3}$$

A key observation in this paper is that many settings are cumbersome to formulate with constraints only, but are more easily described with combinations of constraints and dominance relations. This includes maximal, closed and free itemset mining, relevant subgroup discovery and sky patterns. We will first introduce these problems, where at this moment we closely follow the notation used in the papers that introduced these settings.

a) Maximal Itemset Mining: Maximal frequent itemsets are *maximal* in that there exists no larger itemset that is still frequent:

$$\{X \subseteq I \,|\, |cover_D(X)| \geq \theta \wedge \nexists Y \supset X : cover_D(Y) \geq \theta\} \tag{4}$$

Here, Y *dominates* X iff $Y \supset X \wedge cover_D(Y) \geq \theta$. Observe that $cover_D(Y) \geq \theta$ can be computed independent of X and is hence a local constraint. To mine maximal patterns, we can simply introduce a dominance relation between two patterns X and Y stating that Y *dominates* X iff $Y \supset X$; we are interested in those patterns that are not dominated *within* the set of all itemsets that are frequent.

b) Closed Frequent Itemset Mining: This setting was introduced by Pasquier et al. [12]. It can be formalized as the problem of finding

$$\{X \subseteq I \,|\, |cover_D(X)| \geq \theta \wedge \nexists Y \subseteq I : Y \supset X \wedge cover_D(Y) = cover_D(X)\} \tag{5}$$

Hence Y *dominates* X iff $Y \supset X \wedge cover_D(Y) = cover_D(X)$. In this case, if a solution is not dominated by any other, it is a closed itemset.

c) Free Itemset Mining: Free itemsets are the minimal *generators* of the closed frequent itemsets [13]. The difference with closed itemsets is that we now prefer the smallest subsets among patterns that cover the same transactions. The dominance relation is: Y *dominates* X iff $Y \subset X \wedge cover_D(Y) = cover_D(X)$.

d) Relevant Subgroup Discovery: This example was already mentioned informally in the introduction. The term subgroup discovery or discriminative itemset mining [16] is often used when each transaction in the database has an associated label, for example *positive* or *negative*. The complete definition of relevant subgroup discovery (of which the one in the introduction is a special case) is the following:

$$\{X \subseteq I \,|\, |cover_{D^+}(X)| \geq \theta \wedge \nexists Y \subseteq I : cover_{D^+}(Y) \supseteq cover_{D^+}(X) \wedge cover_{D^-}(Y) \subseteq cover_{D^-}(X) \wedge (cover_D(X) = cover_D(Y) \rightarrow Y \supset X)\} \tag{6}$$

The last condition states that if two patterns cover exactly the same set of transactions, the one with the largest set of items is preferred.

The dominance relation here is that Y *dominates* X iff $cover_{D^+}(Y) \supseteq cover_{D^+}(X) \wedge cover_{D^-}(Y) \subseteq cover_{D^-}(X) \wedge (cover_D(X) = cover_D(Y) \rightarrow Y \supset X)$.

e) Sky Patterns: A last example are the recently introduced sky patterns [14]. The problem of mining sky patterns is similar to that of finding the Pareto-optimal front for a multi-objective optimization problem. More formally, let $m_1(I)$ and $m_2(I)$ be two measures that can be calculated for any itemset

I. For example, m_1 measures the size of the itemset and m_2 measures its frequency. The problem of finding all sky patterns given m_1 and m_2 can be formalized as:

$$\{X \subseteq I \mid |cover_D(X)| \geq 1 \wedge \quad (7)$$
$$\nexists Y \subseteq I : |cover_D(X)| \geq 1 \wedge$$
$$((m_1(Y) > m_1(X) \wedge m_2(Y) \geq m_2(X)) \vee$$
$$(m_1(Y) \geq m_1(X) \wedge m_2(Y) > m_2(X)))\}$$

Here $|cover_D(X)| \geq 1$ is a local constraint; the dominance relation is that Y *dominates* X iff $((m_1(Y) > m_1(X) \wedge m_2(Y) \geq m_2(X)) \vee (m_1(Y) \geq m_1(X) \wedge m_2(Y) > m_2(X))$.

The above examples provide the intuition that a dominance relation captures a wide range of non-local constraints and pattern mining tasks. The problem of how to specify them in a simple general framework is addressed next.

III. An algebra for Dominance Programming

The proposed *algebra* will allow us to specify both the local constraints and the dominance relations of the above problems in a concise way. The algebra consists of two parts: a *constraint algebra*, which will be used to express local constraints, and a *dominance algebra*, which will be used to express the dominance relations; the main novelty in our work is the use of an algebra to combine constraints with dominance relations, and the use of an algebra to specify dominance relations.

A. A Constraint Algebra for Local Constraints

Our approach combines ideas from database theory with ideas from constraint programming. Central is the idea that a local pattern mining problem can be seen as a constraint satisfaction problem (CSP) [7], where each pattern corresponds to a solution of the CSP. More formally, a CSP $\mathcal{P} = (\mathcal{V}, \mathbf{D}, \mathcal{C})$ [17] is specified by

- a finite set of variables $\mathcal{V}$;
- an initial domain $\mathbf{D}$, which maps every variable $v \in \mathcal{V}$ to a finite set of values $\mathbf{D}(v)$;
- a finite set of constraints $\mathcal{C}$.

Solving a CSP corresponds to finding an assignment to the variables in $\mathcal{V}$ from their domain $\mathbf{D}(v)$ such that all constraints in $\mathcal{C}$ are satisfied.

We can represent itemset mining problems with local constraints as constraint satisfaction problems, following the methodology of De Raedt et al. [7]. As an example consider the problem of frequent itemset mining. The problem of frequent itemset mining can be represented by means of two sets of variables:

- $\mathcal{I} = \{i_1, \ldots, i_n\}$, which represent the items;
- $\mathcal{T} = \{t_1, \ldots, t_m\}$, which represent the transactions.

Hence, $\mathcal{V} = \mathcal{I} \cup \mathcal{T}$. All variables $v \in \mathcal{V}$ have a binary domain: $\mathbf{D}(v) = \{0, 1\}$. Finally, we impose the following constraints between these variables:

- *coverage*, stating that a transaction is covered (Equation 2) iff it contains all items in the itemset represented by the item variables. The constraint that should be satisfied between the variables in $\mathcal{I}$ and $\mathcal{T}$ is hence: $cover_D(\mathcal{I}, \mathcal{T}) \equiv (\forall t_j \in \mathcal{T} : t_j = 1 \leftrightarrow (\forall i_k \in \mathcal{I} : (i_k = 1 \rightarrow (j,k) \in D))$;
- minimum frequency, stating that the sum of the transaction variables exceeds a given threshold θ: $support(\mathcal{T}, \theta) \equiv \sum_{t_j \in \mathcal{T}} t_j \geq \theta$.

One way of looking at this, is that from all potential assignments to the variables in $\mathcal{V}$, the constraints select a subset. This is illustrated in Figure 1.

	i_1	i_2
t_1	1	1
t_2	0	1
t_3	0	1

(a) a small database

i_1	i_2	t_1	t_2	t_3
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
⋮	⋮	⋮	⋮	⋮
1	1	1	1	1

(b) all potential assignments for the variables

	i_1	i_2	t_1	t_2	t_3	represented itemset
	0	0	1	1	1	$\emptyset$
$\Rightarrow$	0	1	1	1	1	$\{2\}$

(c) all assignments satisfying the coverage and the minimum support constraint with $\theta = 2$.

Fig. 1. Illustration of the potential solutions selected by a minimum support constraint for a given database.

We can observe a similarity between database querying and constraint satisfaction at this point. If we would have a table with all potential assignments to all variables (hence, we would materialize the table in Figure 1(b), where each column of the table corresponds to a variable in the CSP), we could find the solutions to the itemset mining problem by only *selecting* those rows (assignments) from the table that satisfy the constraints. Our algebra exploits this observation, allowing the reuse of concepts of relational algebra to formalize mining problems.

Expressions in our algebra for specifying constraint satisfaction problems are defined as follows.

Definition 1 (Constraint Algebra)**.** *Expressions in the constraint algebra are inductively defined as follows.*

- *(generator) let a and b be integers, then $\{a, \ldots, b\}$ is an expression in our algebra; it defines a table with a single column of length $|\{a, \ldots, b\}|$ where each row corresponds to a different value from $\{a, \ldots, b\}$*
- *(product) let E_1 and E_2 be expressions in our algebra, then $E_1 \times E_2$ is an expression in our algebra; let T_1 and T_2 be the tables represented by E_1 and E_2, then $E_1 \times E_2$ defines the table $\{(v_1, \ldots, v_n, u_1, \ldots, u_m) \mid (v_1, \ldots v_n) \in T_1, (u_1, \ldots, u_m) \in T_2\}$*
- *(power) let E be an expression in our algebra and let n be an integer, then E^n is an expression in our algebra; let T be the table represented by E, then E^n represents the table $T \times T \times \cdots \times T$, where the product is taken $n-1$ times*
- *(renaming) let E be an expression in our algebra and V an identifier, then $\lambda_V(E)$ is an expression; let T be the table represented by E, then $\lambda_V(E)$ represents the table T in which all columns have been renamed with names*

$V[1]...V[n]$, *where* n *is the number of columns in* T. *If* $n = 1$, *the name is assumed to be* V; V *will be referred to as a* variable name

- *(selection) let* E *be an expression in our algebra and let* c *be a constraint, then* $\sigma_\varphi(E)$ *is an expression in our algebra as well; let* T *be the table represented by expression* E, *then* $\sigma_\varphi(E)$ *represents the table with all rows in* T *that satisfy constraint* φ.

Note that we made a deliberate choice to use a notation which is close that of relational algebra. As an example, we can formalize the problem of frequent itemset mining with the following expression.

$$E_{fi} \equiv \sigma_{support(T,\theta)}\left(\sigma_{cover(I,T)}(\lambda_I(\{0,1\}^n) \times \lambda_T(\{0,1\}^m))\right)$$

The expression reads as follows:

1) The generator $(\lambda_I(\{0,1\}^n) \times \lambda_T(\{0,1\}^m))$ conceptually describes all the tuples $(i_1, \ldots, i_n, t_1, \ldots, t_m)$ in $I^n \times T^m$. Each tuple represents an itemset and a set of transaction identifiers.
2) The inner select operator $(\sigma_{cover(I,T)}(\ldots))$ selects only the tuples in which a value $t_i = 1$ iff the transaction t_i covers the itemset $(i_1, \ldots, i_n)$. These are all the tuples that represent itemsets and their corresponding cover.
3) The outer select operator $(\sigma_{support(T,\theta)}(\ldots))$ selects only the tuples in which the number of t_is equal to 1 is greater than θ. In other words, all the tuples that represent frequent itemsets and their cover.

The selection operator in our algebra can in principle use all constraints available in traditional constraint programming systems [17]. Indeed, one can see that expressions in the constraint algebra correspond closely to the basic elements of a CSP: the λ operator essentially introduces variables, while the σ operator introduces constraints between these variables. Consequently, an expression in the constraint algebra could rather straightforwardly be evaluated using generic constraint programming systems, as was shown in [18].

B. A Dominance Algebra for Programming Pre-orders

We will now discuss how the dominance relationship introduced in Section II can be formalized in an extension of the constraint algebra. The main idea is to express the domination relations as pairwise preferences between assignments to variables. The main theoretical tool is that of preorders.

Definition 2. *Let* P *be a set and let* R *be a binary relation over elements in* P, *i.e.* $R \subseteq P \times P$; *then* R *is a preorder if:*

- *(transitivity) if* xRy *and* yRz, *then* xRz;
- *(reflexivity) for all* $x \in P$: xRx.

Here xRy *is a shorthand for* $(x, y) \in R$.

In our case, the set P will be a set of solutions to a CSP, i.e. P is the set of all rows in a table T defined by an expression E in the constraint algebra.

For a given preorder, we can now define the *dominance operator* in our algebra:

Definition 3 (Dominance Operator)**.** *Let* E *be an expression in the constraint algebra and let* T *be the table represented by the expression* E. *Furthermore, let* R *be a preorder over the elements in* T. *Then* $\sigma_R(E)$ *represents the following table:*

$$\sigma_R(T) = \{\vec{x} \in T \mid \forall \vec{y} \in T : \vec{y}R\vec{x} \rightarrow \vec{x}R\vec{y}\}$$

i.e., the set of all rows that are not strictly dominated according to the preorder, that is, they are only dominated by equivalent solutions.

As an example, consider $T = \{a, b, c, d\}$ and $R = \{(a, a), (b, b), (c, c), (d, d), (a, b), (b, a), (c, d)\}$. According to the dominance relation R, a and b are equivalent (because (a, b) and (b, a) are both in R) but incomparable to either c or d. Moreover, d is dominated (but not equivalent) to c. We thus have $\sigma_R(T) = \{a, b, c\}$.

The main remaining question is now how to specify the preorder R. Clearly an extensional definition of R is not practical when the number of tuples is large; therefore we introduce an algebra for programming preorders.

Definition 4 (Preorder Algebra)**.** *Expressions* E *in the preorder algebra, for a table* T *with column names* V, *are inductively obtained as follows:*

- *let* v *be a variable (column) in* V, *then* $(\geq_v)$, $(\leq_v)$ *and* $=_v$ *are expressions in the preorder algebra. They define preorders*

  $$(\geq_v) \equiv \{(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in T, x_v \geq y_v\}$$

  and

  $$(\leq_v) \equiv \{(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in T, x_v \leq y_v\};$$

  finally,

  $$(=_v) \equiv \{(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in T, x_v = y_v\};$$

  here x_v *and* y_v *denote the values of variable* v *in tuple* $\vec{x}$ *and* $\vec{y}$, *respectively; note that the values each variable* v *can take are assumed to be ordered;*
- *let* E_1 *and* E_2 *be expressions in the preorder algebra, then* $E_1 \wedge E_2$ *is an expression in the preorder algebra as well; let* R_1 *and* R_2 *be the preorders identified by these expressions, then* $E_1 \wedge E_2$ *defines a preorder*

  $$(\vec{x}, \vec{y}) \in R_1 \wedge R_2 \Leftrightarrow (\vec{x}, \vec{y}) \in R_1 \wedge (\vec{x}, \vec{y}) \in R_2.$$

Let us consider the example of maximal frequent itemset mining to illustrate this algebra. Remember from Section II that the problem can be formalized as follows:

$$\{X \subseteq I \mid |cover_D(X)| \geq \theta \wedge \tag{8}$$

$$\nexists Y \supset X : cover_D(Y) \geq \theta\} \tag{9}$$

with the following dominance relation: Y *dominates* X iff $Y \supseteq X$. In our tabular representation of potential solutions, this means that a row $\vec{x}$ representing one pattern, dominates another row $\vec{y}$ representing another pattern, iff for each column v representing an item, the inequality $x_v \geq y_v$ holds. We can formalize this preorder by this expression:

$$\wedge_{i \in I}(\geq_i),$$

where I is the set of columns corresponding to item variables.

To obtain the maximal frequent itemset mining problem, we can now formulate this dominance relation in our algebra, and apply it to the expression representing all frequent itemsets (E_{fi}): $\sigma_{\wedge_{i\in I}(\geq_i)}(E_{fi})$. In total we now have the following expression:

$$\sigma_{\wedge_{i\in I}(\geq_i)}(\sigma_{support(T,\theta)}(\sigma_{cover(I,T)}(\lambda_I(\{0,1\}^n)\times\lambda_T(\{0,1\}^m))))$$

For example, in Figure 1 the final set of patterns determined by this expression would consist of a single row $(\mathbf{0},\mathbf{1},1,1,0)$ as it dominates the row $(\mathbf{0},\mathbf{0},1,1,1)$ (I variables indicated in bold).

We believe that this closely corresponds to the intuition most researchers have about this problem: among all frequent itemsets, we are interested in finding only those that are maximal.

IV. Examples of Dominance Programming

We now show how the examples in Section II, as well as others, can be expressed in the algebra.

A. Closed Frequent Itemset Mining

The expression for this problem extends that of the frequent itemset mining problem. In addition, we need to express that if two itemsets cover the same set of transactions, we prefer the larger one. We can formalize this in the preorder algebra with the following expression: $(\wedge_{t\in T}(=_t))\wedge(\wedge_{i\in I}(\geq_i))$. The closed frequent itemset mining problem is then expressed in total with:

$$\sigma_{(\wedge_{i\in I}(\geq_i))\wedge(\wedge_{t\in T}(=_t))}(E_{fi})$$

B. Free Itemsets

From a dominance point of view, the only difference with closed itemsets is that now **subsets** dominate the supersets covering the same transactions: $(\wedge_{t\in T}(=_t))\wedge(\wedge_{i\in I}(\leq_i))$. Free frequent itemset mining can hence be expressed as:

$$\sigma_{\left(\wedge_{i\in I}(\leq_i)\right)\wedge(\wedge_{t\in T}(=_t))}(E_{fi})$$

C. Cost-based Itemset Mining

We now first demonstrate how additional *local* constraints can be expressed in our algebra. In the next section, we will extend this formulation with dominance relations. A prototypical local constraint is a constraint on the *cost* of an itemset, assuming every item has an individual cost [4]. Given a cost vector $C=(c_1,\ldots,c_n)$ that contains a cost c_i for every item i, the cost of an itemset I can be computed as follows: $cost(I)=\sum_{i\in I}C_i$. Using our algebra, we extend the standard frequent itemset mining expression with a variable c and constrain it to the cost of the itemset:

$$E_{fic}\equiv\sigma_{c=cost(I)}(E_{fi}\times\lambda_c(\{0,\ldots,n\})) \tag{10}$$

D. Cost-based Itemset mining and Dominance Relations

As studied by [10], combining closed itemset mining with cost-based itemset mining with a maximum cost can result in different solutions depending on the interpretation: one can either mine all closed itemsets and filter out the ones with a too high cost (as one would do in post-processing), or calculate the closure of all itemsets that have a cost lower than some threshold. While the former is typically implemented in existing systems (out of practical reasons), the latter is actually more meaningful.

Our algebra allows to express both variants. Let $\sigma_{clo}\equiv\sigma_{(\wedge_{t\in T}(=_t))\wedge(\wedge_{i\in I}(\geq_i))}$, then the closed itemsets that are not too costly are formalized as:

$$\sigma_{c\leq\theta}(\sigma_{clo}(E_{fic})); \tag{11}$$

and the closure over the itemsets that are not too costly:

$$\sigma_{clo}(\sigma_{c\leq\theta}(E_{fic})). \tag{12}$$

This demonstrates that the algebra is rich enough to cover settings that have been problematic in the standard constraint-based mining framework up to now and also does this in an intuitive way.

E. Sky Patterns

We illustrate the sky pattern setting on an example by Soulet et al. [14]. They address the problem of extracting sky patterns with respect to the *frequency* and *area* measures. The frequency of an itemset is the size of its cover: $freq(I)=|cover(I)|$, while the area of an itemset corresponds to the dimension of the itemset in terms of items and transactions: $area(I,T)=|\{(i,t)|i\in I,t\in T\}|=|I|*|T|$.

We can reuse the expression for frequent itemset mining here, with the assumption that $\theta=1$.

Next, as was the case for cost-based itemset mining, we add two integers f and a representing the frequency and the area respectively:

$$E_{i2}\equiv\sigma_{f=freq(I)}(E_{fi}\times\lambda_f(\{0,\ldots,m\})) \tag{13}$$

$$E_{i3}\equiv\sigma_{a=area(I,T)}(E_{i2}\times\lambda_a(\{0,\ldots,(n*m)\})) \tag{14}$$

Then mining the sky itemsets with respect to the set of measures $\{frequency,area\}$ can be formalized as follows:

$$\sigma_{(\geq_a)\wedge(\geq_f)}(E_{i3}) \tag{15}$$

F. Relevant Subgroup Discovery

The setting of relevant subgroup discovery introduced earlier can be expressed using the following expression:

$$\sigma_{\left(\wedge_{t\in T^+\cup T^-}(=_t)\right)\wedge(\wedge_{i\in I}(\geq_i))}\left(\sigma_{\left(\wedge_{t\in T^+}(\geq_t)\right)\wedge\left(\wedge_{t\in T^-}(\leq_t)\right)}\left(\sigma_{cover(I,T^+)}(\lambda_{T^+}(\{0,1\}^{m^+})\times\sigma_{cover(I,T^-)}(\lambda_{T^-}(\{0,1\}^{m^-})\times\lambda_I(\{0,1\}^n)))\right)\right)$$

In this expression, the inner-most dominance operator preserves patterns with the same transaction set because they are considered equivalent. The second operator ensures that among such patterns the largest ones are preferred, as required.

G. Dominated Patterns in PN Space

Relevant subgroups dominate each other based on the positive and negative transactions covered. Instead, one could also impose that a pattern dominates another pattern if it simply covers more (or equal) positive transactions and less (or equal) negative transactions. The problem is similar to that of finding all patterns on the convex hull in PN space [15] and is related to that of finding sky patterns.

In our algebraic notation, this can be expressed by introducing integers p and q that represent the number of positive/negative transactions covered, and imposing the above mentioned dominance relation:

$$\sigma_{(\geq_p)\wedge(\leq_q)}(\sigma_{p=freq^+(I)\wedge q=freq^-(I)}(\\ \lambda_I(\{0,1\}^n) \times \lambda_p(\{0,\ldots,m^+\}) \times \lambda_q(\{0,\ldots,m^-\})))$$

H. Novel settings

The algebra is not restricted to formulating existing problems. It provides a general yet well-founded way to express any combination of local constraints and domination relations.

For example, one can formulate the problem of finding the smallest maximal itemsets as follows:

$$\sigma_{\leq_s}(\sigma_{\wedge_{i\in I}(\geq_i)}(\sigma_{s=|I|}(E_{fi} \times \lambda_s(\{0,\ldots,n\}))))). \quad (16)$$

As far as we now, there exists no algorithm capable of addressing this simple problem.

While the above has shown how a wide range of mining tasks can be formulated using the algebra, more than is possible in existing constraint-based mining frameworks, we will now explain a general way to solve problems expressed in this algebra.

V. Evaluating Expressions in the Dominance Algebra

To evaluate expressions formalized in the dominance algebra, we first show that every expression can be reduced to a normal form. Hence, it suffices to design an approach to evaluate expressions in normal form. Building on earlier work [18], we then propose to evaluate expressions using constraint programming techniques. We will give a brief introduction to constraint programming systems; then we will discuss the modifications needed to handle the dominance algebra.

A. Normal Form

We can observe the following property on the product operator $\times$ in our algebra.

Lemma 1. *Given two dominance algebraic expressions E_1 and E_2, $\sigma(E_1) \times E_2$ is equivalent to $\sigma(E_1 \times E_2)$, i.e. both expressions define the same solution set, where σ is either a dominance operator or a selection operator that only uses variables present in E_1.*

Proof: For the selection operator this property carries over straightforwardly from a similar property for the relational algebras in database theory. For the dominance operator we can prove the two directions: if $(\vec{x},\vec{y}) \in \sigma_R(E_1) \times E_2$, then $\vec{x} \in \sigma_R(E_1)$; then all solutions $(\vec{x},\vec{y})$ with the same $\vec{x}$ but different $\vec{y}$ must be equivalent under R, as R does not depend on the variables in $\vec{y}$. The operator σ_R returns all equivalent solutions, and hence $(\vec{x},\vec{y}) \in \sigma_R(E_1 \times E_2)$. The other direction can be shown using similar properties. ■

As a consequence of this lemma, we can always rewrite an expression in the dominance algebra in a normal form of the following kind:

$$\sigma_1(\sigma_2(\cdots\sigma_n(\lambda_{v_1}(\{1,\ldots,c_1\})\times\cdots\times\lambda_{v_m}(\{1,\ldots,c_m\}))\cdots))$$

where $\sigma_1,\ldots,\sigma_{n-1}$ are either dominance or selection operators, and σ_n is a selection operator. Using Lemma 1 one can see that any product of two expressions can always be rewritten by pushing one of the operands of the product deeper in the expression, unless both expressions introduce variables.

We will use constraint programming systems to process expressions in this normal form.

Algorithm 1 Constraint-Search(Domain: D)

```
1: D :=Propagate(D, φ)
2: if constraints were violated then
3:     return
4: end if
5: if ∃x ∈ V : |D(x)| > 1 then
6:     x :=Select-Variable(V)
7:     for all d ∈ D(x) do
8:         Constraint-Search(D ∪ {x ↦ {d}})
9:     end for
10: else
11:     Output solution
12: end if
```

B. Constraint Programming

Constraint programming (CP) systems are general systems for solving constraint satisfaction problems (CSP). Thus these systems can be used to evaluate the constraint algebra defined in Section III-A. An expression of the form $\sigma_\varphi(\lambda_{v_1}(\{1,\ldots,c_1\}) \times \cdots \times \lambda_{v_m}(\{1,\ldots,c_m\}))$, where σ_φ is a selection operator, can easily be evaluated by a constraint programming system by entering all defined variables and all constraints in φ as constraints in the CP system. This is possible for all the constraints studied in this paper.

Algorithm 1 gives a high-level overview of a CP system. Essentially, a CP system is a depth-first search algorithm. The system maintains a domain of potential values for each variable in the CSP; $D(x)$ denotes the possible values that a variable x can still take. Then the CP system searches for assignments of the variables that satisfy all the constraints simultaneously by shrinking the domains of the variables. To shrink the domain $D(x)$ of a variable x, the CP system uses two mechanisms: *constraint propagation* and *search*.

Constraint propagation takes a (partial) solution and evaluates all the constraints, which are stored in a global constraint store φ (line 1). Propagation can have several effects: it can detect failure (i.e. the current partial solution can never be extended to a full solution), it can remove a constraint from consideration, or it can shrink the domain of variables. For

example, consider we have a constraint on three boolean variables that states that $a \vee b \vee c = 1$. Given a partial solution $(1, ?, ?)$, propagation will detect that the constraint is satisfied and can be removed; given a partial solution $(0, 0, ?)$, we can detect that c must be 1. For each constraint, a CP system includes built-in algorithms to perform these reasoning steps.

When no more propagation is possible, the constraint solver invokes the search procedure. The search procedure selects an unassigned variable (line 6) according to a user-defined heuristic (for example the variable with the smallest domain) and assigns it with a value (line 7). The order in which variables are selected is known as *variable ordering*; the order in which values are selected is known as *value ordering*. The value and variable ordering do not affect the correctness of the algorithm, but may affect its efficiency.

After the assignment, the solving procedure is called recursively (line 8).

C. Evaluating Dominance Expressions with CP

Using a constraint programming system such as the one shown in Algorithm 1, we can employ the following straightforward strategy for evaluating any expression in the normal form:

1) run a CP system to evaluate $\sigma_{\varphi}(\lambda_{v_1}(\{1, \ldots, c_1\}) \times \cdots \times \lambda_{v_m}(\{1, \ldots, c_m\}))$ and store the result set;
2) post-process the resulting set of solutions iteratively by applying implementations of $\sigma_{n-1}, \ldots \sigma_1$ consecutively. The implementations take a set as input and remove all the non-dominating solutions.

While correct, this approach is not very efficient: the first step generates the complete set of patterns satisfying the constraints, even if the dominance relation is likely to eliminate most of them during the post-processing step. Maintaining and post-processing a large number of intermediate results is computationally expensive. In order to reduce the size of the intermediate solution step, we combine two strategies: (1) update the constraint satisfaction problem to eliminate unwanted solutions from the unexplored search space and (2) influence the value ordering in order to maximize the impact of strategy (1).

The main idea is to use the inner-most dominance operator $\sigma_{n-1} = \sigma_R$ to guide and constrain the search of the CP system. To this end, we modify Algorithm 1 such that, instead of just outputting a solution for post-processing (line 11), we also update the set of constraints such that the CP system will avoid producing any solutions dominated by the current solution. In general, we can represent a dominance relation R as

$$\bigwedge_{v \in V} (\geq_v) \wedge \bigwedge_{w \in W} (\leq_w).$$

For each solution D, outputted by Algorithm 1, it therefore suffices to add the constraint

$$\bigvee_{v \in V} (v > D(v)) \vee \bigvee_{w \in W} (w < D(w)) \vee \bigwedge_{v \in V \cup W} (v = D(v)),$$

which states that the future solution is either not dominated by D, or that it is equivalent with it (with respect to the variables used in the dominance relation).

By employing this strategy, the CP system will avoid generating solutions that do not satisfy the dominance relation. However, this strategy can only discard solutions that have not been generated yet. In order to maximize the effectiveness of this strategy, we should therefore also take care of selecting a search order in which solutions are produced in the most beneficial order. In the case of a single dominance operator, we can always derive an *optimal* search order for which we can guarantee that any solution produced by the modified Algorithm 1 satisfies the dominance relation, thus eliminating the need for post-processing. For each variable v in the dominance relation R, we simply assign values from smallest to largest (in case of $\leq v$) or largest to smallest (in case of $\geq v$). (Note that the order in which we select the variables is irrelevant, only the order in which we select the values for each variable matters.)

In the general case of multiple stacked dominance operators, it is often impossible to find such an optimal order and we need to use post-processing. In this case, it suffices to enumerate all solutions in reverse order and apply the same constraint update procedure as before (but replacing the CP search algorithm by an enumeration of the intermediate solutions).

It is also important to note that the search order can also significantly impact the efficiency of the constraint programming system, such that the optimal search order for the dominance relation may not be the same as the (often unknown) optimal order for solving the underlying constraint satisfaction problem. In some cases, it may therefore be beneficial to diverge from the optimal search order and use one that is better suited for the underlying CSP. Examining this trade-off is an important part of future work.

VI. Experiments

In this section we evaluate our generic dominance programming approach on several tasks. We have implemented dominance programming by extending the state-of-the-art Gecode constraint solving system [19], version 4.0.0. The implementation is available on the authors' website. The formulations for the local constraints were taken from CP4IM [18]. In the following experiments, we compare with ACminer (v1.0) [13] and Eclat as implemented by Borgelt (v4.0) [20]; LCM (v5.3) [21] and CP4IM (v3.7.3) [18]; and Aetheris (v0.0.2) [14]. All datasets come from the *UCI machine learning repository* and were obtained online[1]. A description of the datasets can be found in [18]. Unless mentioned otherwise, the datasets plotted show representative results. The experiments were run on computers running Ubuntu 12.04 with quad-core Intel i7 processors and 16Gb of ram.

A. Closed and free itemset mining

As a baseline comparison, we compare our system with state-of-the-art systems on the task of closed and free frequent

[1] http://dtai.cs.kuleuven.be/CP4IM/datasets/

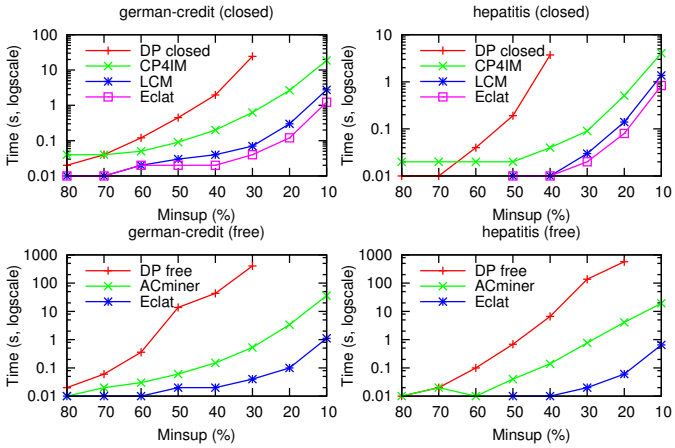

Fig. 2. Enumerating closed and free itemsets. Free itemsets mining (top), closed itemset mining (bottom).

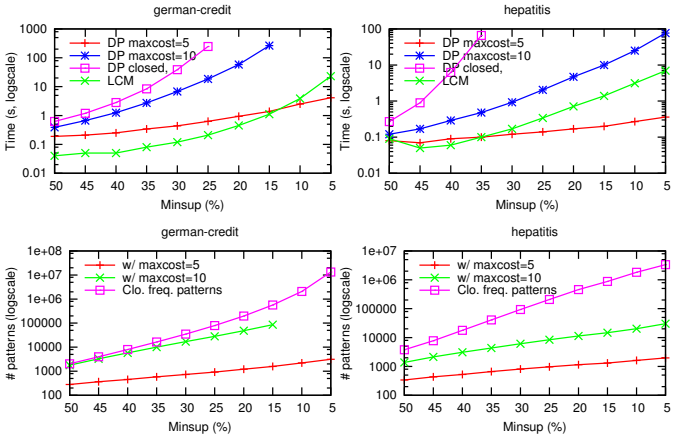

Fig. 3. Enumerating the closed set of frequent patterns with cost constraints: runtimes (top), number of patterns (bottom).

itemset mining. Note that we do not have the ambition to be faster on these specific well-studied problems. Indeed, it is not expected that a constraint programming approach is faster on these tasks, as can be seen by comparing the runtime of the CP-based CP4IM system with Eclat and LCM in Figure 2 (top). Furthermore, closed frequent itemsets can be enumerated in polynomial delay while our system adds constraints for every solution found. However, such optimizations do not carry over to other settings, while our system can handle a large number of settings in a generic way, as we will see.

Figure 2 (bottom) shows similar results for free frequent itemset mining. Note that the general CP4IM system cannot handle this mining task in a simple way, while ours can.

B. Combining closed and cost constraints

As explained in [10] and mentioned earlier, combining the closedness constraint with a minimum frequency and maximum cost constraint can be done in two different ways. The first formulation (Equation 12) represents the naive but algorithmically simpler interpretation, while the second formulation (Equation 11) represent the more meaningful interpretation of taking the closure of all itemsets with a cost lower than some threshold.

Both can be formulated in our framework, and we report only on the second formulation. We used a unary cost for each item and threshold it at 10, respectively 5, percent of the maximum size. Figure 3 (top) shows runtimes for the two settings, while the bottom figures show the number of patterns. The runtime of LCM represents the minimum amount of time needed in case one would post-process all closed patterns. We can observe that a higher threshold leads to fewer patterns and lower runtimes, indicating the effectiveness of the search procedure. This is most obvious for low minimum support and cost thresholds, where the search is much more efficient than a post-processing approach would be.

C. Relevant subgroup discovery

We next compare our system on the task of relevant subgroup discovery to the proposed approach of Garriga et al [9]. They propose a two-step post-processing approach [9]:

1) extract the set of all frequent closed patterns $\mathcal{P}^+$, on the dataset with positive transactions;
2) post-process this set by removing every pattern $X \in \mathcal{P}^+$ that has a subset $Y \subseteq X$ with the same cover in the negative transactions.

Due to the lack of implementation and as advised by the original authors, we used LCM for step one and implemented a post processor in C++ for step 2.

Figure 4 shows the runtime (top) and number of solutions (bottom). For most of the datasets of the UCI repository, the number of closed-on-the-positive patterns is close to the number of relevant subgroups (such as in the *german-credit* dataset). In such cases, almost no additional pruning can be done and the post-processing approach is the most efficient. However, when the number of closed patterns diverges from the number of subgroups (such as in the *hepatitis* dataset), the post-processing approach has to handle an overwhelming number of closed patterns and thus performs poorly. In contrast, our approach performs efficiently because it can prune false-positive candidate patterns (i.e. candidate patterns that have subsets in the negative transactions).

D. Sky patterns

Finally, we compare our system to the specialized sky pattern mining system Aetheris [14]. The task is to find all sky patterns according to the frequency and area measures, as explained in Section IV. Following the experimental protocol in [14], we also add a minimum support threshold to better study the behavior of the systems.

It is worth mentioning that for this problem the branching strategy was to select the smallest itemsets first (based on the observations made by [18]). This branching strategy is not optimal in the sense that it requires post-processing but is more efficient in practice.

Figure 5 (top) shows that for decreasing minimum support thresholds, our system is increasingly more successful in efficiently pruning the search space compared to Aetheris. The bottom figures show that a post-processing approach would not

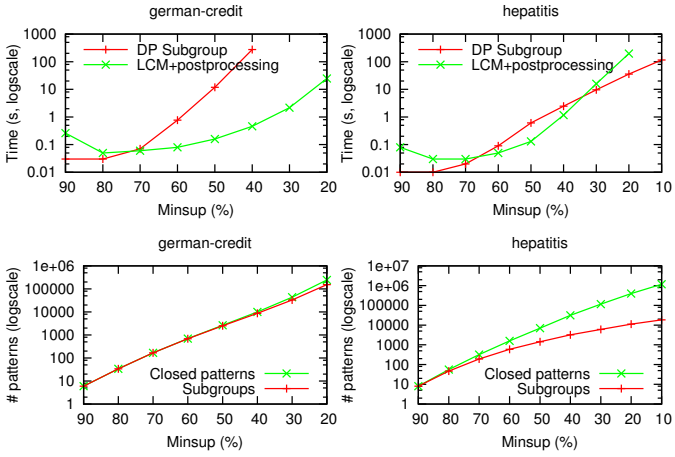

Fig. 4. Enumerating relevant subgroups: runtimes (top), number of patterns (bottom).

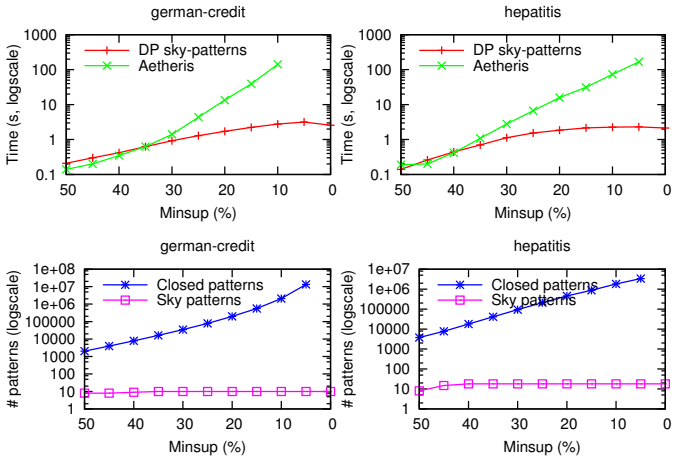

Fig. 5. Enumerating sky patterns (freq + area): runtimes (top), number of patterns (bottom).

be feasible for low thresholds as the difference between the number of closed patterns and sky patterns increases rapidly (note the exponential scale). On the contrary, our system is not impacted by the number of intermediate frequent closed patterns and is thus able to mine datasets without the frequency constraint (i.e. with $\theta = 1$).

VII. Related Work

The approach presented in this paper has several key features, which have been studied individually in the past:

Constraint programming: As discussed earlier in detail, our approach extends the work of De Raedt et al. [7], which showed that constraint programming is an effective and generic paradigm to address and solve constraint based itemset mining problems.

Constraint programming has also been proposed as a solution for the problem of k-pattern set mining [22]. However, in this work a fixed size k of the output was assumed. The algebraic approach presented in this paper does not assume a fixed pattern set size and is more scalable.

Within the constraint programming literature, our approach is related to *CP-Nets* [23]. CP-Nets provide a generic approach for specifying preference relations between solutions; they can be seen as an alternative formalism for specifying dominance relations. However, our algebra is more practical for the specification of orders in pattern mining.

Generic pattern mining and condensed representations: A class of well-known generic methods are those that search for borders in version spaces under monotonic and anti-monotonic constraints [24], [25]. Our work is different in several ways. First, the constraints in our framework are not necessarily monotonic or anti-monotonic. Second, we rely on generic constraint solving technology to support this wide range of constraints. Moreover [26], proposes an algorithm that can address a broader range of constraints, but do not provide a language to describe them. Our dominance algebra represents a very different approach to problem formalization.

Most other generic approaches focus only on local constraints; they do not take into account relationships between patterns and do not build on constraint satisfaction technology [4], [6].

Multi-objective optimization: The framework that we propose in this paper is closely related to multi-objective optimization problems (MOOP) [27], and the identification of Pareto optimal sets [28]. Our dominance algebra puts a much stronger focus on expressing dominance relations and clarifies the relationships between MOOPs and itemset mining problems. Furthermore, our dominance algebra is explicitly developed to support preorders over large numbers of variables. In practice, the number of variables over which a dominance is defined in MOOPs is typically smaller.

Skyline patterns and queries: The work by Soulet et al. on skyline patterns [14] can be seen as a direct application of the MOOP framework to pattern mining. It assumes that an order is defined over a small number of scoring functions and does not support the orders that are needed to deal with problems such as relevant pattern mining and free itemset mining. The setup of Soulet et al. however fits nicely in the dominance programming framework.

The methodology presented in this paper has clear relationships to methodologies developed in the database community. Dominance reporting is a problem also relevant to the database community [29]. Skyline queries have been developed to deal with dominance relationships in databases [30]. Our work is different from traditional skyline queries as it deals with pattern mining problems where a combinatorial search is needed. Our algebraic notation closely resembles that of Codd's relational algebra [31], but applies this notation in a context where combinatorial search is needed.

VIII. Conclusions

In this paper, we have observed that dominance relations can be found in many pattern mining settings. Building on this observation, we have proposed an algebra that combines constraints and dominance relations and that can be used to adequately describe a broad range of pattern mining settings. This algebra resembles relational algebras and arguably would be easy to integrate in a database system.

To evaluate expressions in our algebra, we have proposed a methodology based on the constraint programming technology. Despite the gain in generality provided by dominance programming, our system can compete with specialized mining algorithms and even outperform them in some cases. We believe that this is a strong indication that dominance programming uses the right concepts to describe pattern mining tasks.

Because of its explanatory nature, this work leaves a number of open questions:

Query rewriting: Given the close connection between data mining and databases it is natural to wonder whether common query optimization techniques in databases can also be applied to dominance programming.

Advanced data structures for evaluating dominance: Specialized algorithms for dominance reporting [29] could be used transparently by the CP system to improve efficiency. Similarly, optimized data structures for checking the dominance within a set of item sets may be used as well.

Intelligent variable and value ordering: At the moment, our system selects a value ordering that eliminates the need for post-processing. This order may not always be the most efficient to solve the core CSP. Studying the impact of the value ordering on the time required to evaluate dominance programs would also help to improve the efficiency.

Furthermore, this paper has a strong focus on itemset mining. An interesting question is how our algebra can be used to formulate more structured mining tasks such as sequence or graph mining. Furthermore, there is no reason to believe that dominance programming could not be used for other types of problems such as resource allocation problems.

ACKNOWLEDGMENTS

We would like to thank the authors of ACminer and Aetheris for sending us their code, and the authors of Eclat, LCM, and CP4IM for making their code available online. This work was supported by the European Commission under the project "Inductive Constraint Programming" contract number FP7-284715, by the Research Foundation–Flanders by means of two Postdoc grants and by the project "Principles of Patternset Mining".

REFERENCES

[1] H. Mannila and H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," *Data Mining & Knowledge Discovery.*, vol. 1, no. 3, pp. 241–258, 1997.

[2] B. Crémilleux and A. Soulet, "Discovering knowledge from local patterns with global constraints," in *Parallel Universes and Local Patterns*. Internationales Begegnungs- und Forschungszentrum fur Informatik, Schloss Dagstuhl, Germany, 2007.

[3] R. Chan, Q. Yang, and Y.-D. Shen, "Mining high utility itemsets," in *3rd Internation Conference on Data Mining*. IEEE, 2003, pp. 19–26.

[4] J. Pei, J. Han, and L. V. Lakshmanan, "Mining frequent itemsets with convertible constraints," in *17th International Conference on Data Engineering*. IEEE, 2001, pp. 433–442.

[5] S. Morishita and J. Sese, "Traversing itemset lattice with statistical metric pruning," in *19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 2000, pp. 226–236.

[6] F. Bonchi and C. Lucchese, "Extending the state-of-the-art of constraint-based pattern discovery," *Data & Knowledge Engineering*, vol. 60, no. 2, pp. 377–399, 2007.

[7] L. De Raedt, T. Guns, and S. Nijssen, "Constraint programming for itemset mining," in *14th SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 204–212.

[8] N. Lavrač and D. Gamberger, "Relevancy in constraint-based subgroup discovery," in *Constraint-Based Mining and Inductive Databases*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 3848, pp. 243–266.

[9] G. C. Garriga, P. Kralj, and N. Lavrač, "Closed sets for labeled data," *The Journal of Machine Learning Research*, vol. 9, pp. 559–580, 2008.

[10] F. Bonchi and C. Lucchese, "On closed constrained frequent pattern mining," in *4th International Conference on Data Mining*. IEEE, 2004, pp. 35–42.

[11] R. Bayardo, "Efficiently mining long patterns from databases," in *Special Interest Group on Management of Data*. ACM, 1998, pp. 85–93.

[12] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *International Conference on Database Theory*. Springer, 1999, pp. 398–416.

[13] J.-F. Boulicaut and B. Jeudy, "Mining free itemsets under constraints," in *5th International Symposium on Database Engineering & Applications*. IEEE, 2001, pp. 322–329.

[14] A. Soulet, C. Raïssi, M. Plantevit, and B. Crémilleux, "Mining dominant patterns in the sky," in *11th International Conference on Data Mining*. IEEE, 2011, pp. 655–664.

[15] S. Nijssen, T. Guns, and L. De Raedt, "Correlated itemset mining in ROC space: A constraint programming approach," in *15th International Conference on Knowledge Discovery and Data Mining*, P. Flach and M. Zaki, Eds. ACM, Jun. 2009, pp. 647–656.

[16] P. K. Novak, N. Lavrač, and G. I. Webb, "Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining," *Journal of Machine Learning Research*, vol. 10, pp. 377–403, 2009.

[17] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence, 2006.

[18] T. Guns, S. Nijssen, and L. De Raedt, "Itemset mining: A constraint programming perspective," *Artificial Intelligence*, vol. 175, no. 12-13, pp. 1951–1983, 2011.

[19] Gecode Team, "Gecode: Generic constraint development environment," 2010, available from `http://www.gecode.org`. [Online]. Available: http://www.gecode.org

[20] C. Borgelt, "Efficient implementations of Apriori and Eclat," in *Workshop of Frequent Item Set Mining Implementations (FIMI)*, 2003.

[21] T. Uno, M. Kiyomi, and H. Arimura, "Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining," in *1st International Workshop on Open Source Data Mining*. ACM, 2005, pp. 77–86.

[22] T. Guns, S. Nijssen, and L. De Raedt, "k-Pattern set mining under constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 2, pp. 402–418, Feb. 2013.

[23] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole, "CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements," *J. Artif. Intell. Res. (JAIR)*, vol. 21, pp. 135–191, 2004.

[24] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm, "Discovering all most specific sentences," *Transasction on Database Systems*, vol. 28, no. 2, pp. 140–174, 2003.

[25] L. De Raedt, M. Jaeger, S. D. Lee, and H. Mannila, "A theory of inductive query answering," in *2nd International Conference on Data Mining*, 2002, pp. 123–130.

[26] B. Negrevergne, A. Termier, M.-C. Rousset, and J.-F. Mehaut, "ParaMiner: a Generic Pattern Mining Algorithm for Multi-Core Architectures," *Journal of Data Mining and Knowledge Discovery (DMKD)*, 2013, advance online publication. doi 10.1007/s10618-013-0313-2.

[27] L. T. Bui and S. Alam, *Multi-objective optimization in computational intelligence: theory and practice*. Information Science Reference, 2008.

[28] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maximal of a set of vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[29] Q. Shi and J. JáJá, "Fast algorithms for 3-d dominance reporting and counting," *International Journal of Foundations of Computer Science*, vol. 15, no. 4, pp. 673–684, 2004.

[30] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Special Interest Group on Management of Data*, 2003, pp. 467–478.

[31] E. F. Codd, "Relational completeness of data base sublanguages," in *Database Systems*. Prentice-Hall, 1972, pp. 65–98.